SKYJO

SAE201.2024.42

Diagramme UML Analyse :

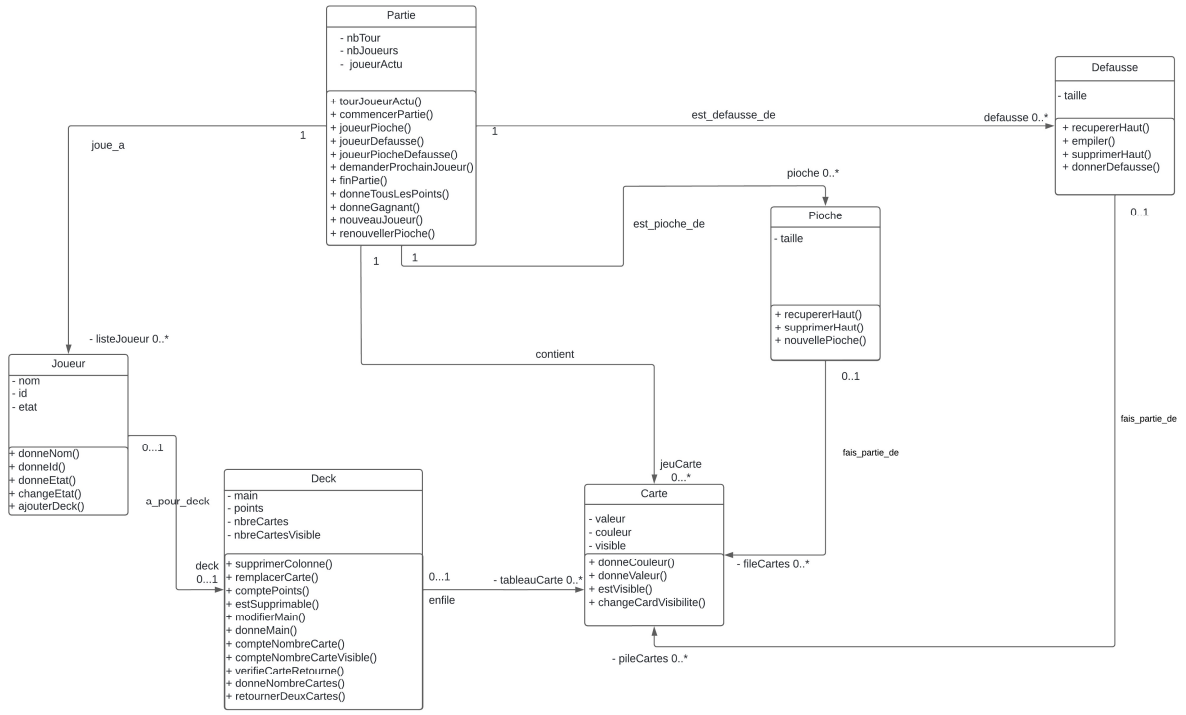
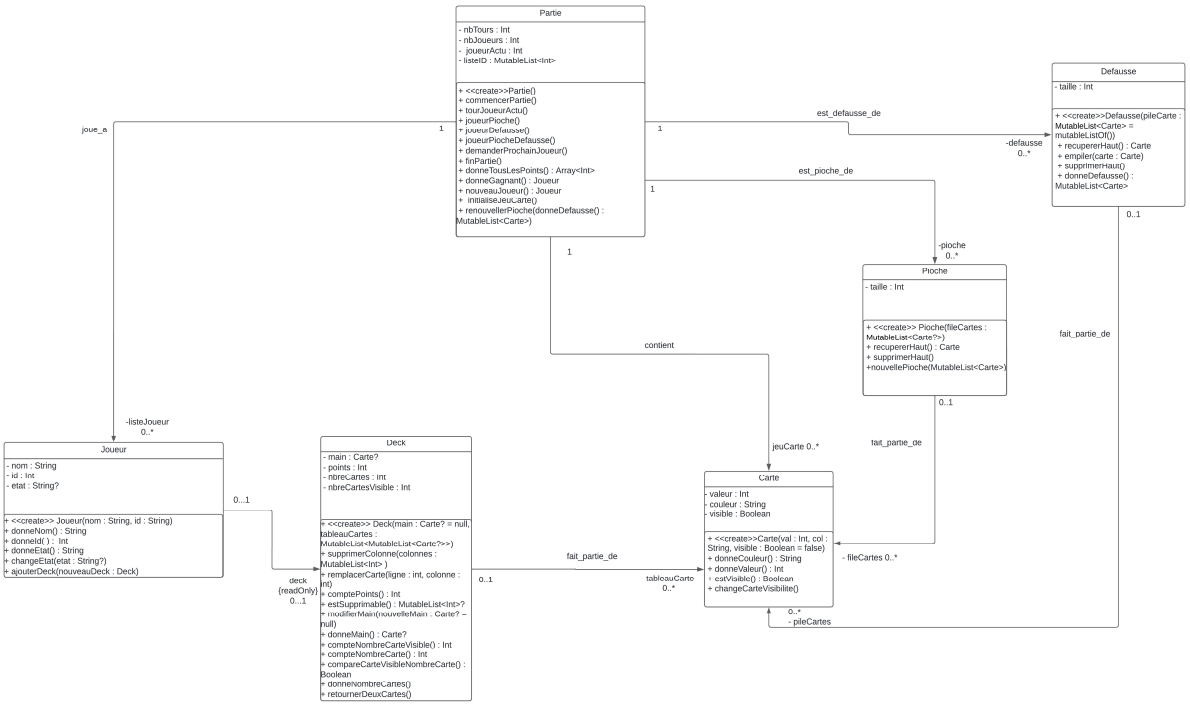


Diagramme UML Conception :



Classe Partie:

Attributs:

- **nbTour** (Int) : permet de renseigner le numéro du tour actuel.
- **nbJoueurs** (Int) : renseigne le nombre de joueurs jouant dans la partie.
- **listeJoueur** (MutableList<Joueur>) : MutableList de type Joueur.
- **joueurActu** (Int) : correspond à l'indice du joueur pour lequel c'est le tour. On aura accès au joueur grâce à la liste `listeJoueur`.
- **jeuCarte** (MutableList<Carte>) : correspond à la liste qui contient toutes les cartes du jeu à l'initialisation de la partie et qui permettra de faire une distribution de ces cartes auprès des decks des joueurs.
- **pioche** (MutableList<Carte>) : c'est la pioche du jeu.
- **defausse** (MutableList<Carte>) : c'est la défausse du jeu.

Méthodes:

- **commencerPartie()** : cette méthode permet d'initialiser la pioche et la défausse.
- **tourJoueurActu()** : cette méthode correspond au tour du joueur, elle permet de mettre en place les différentes actions du joueur possibles (piocher dans la pioche, piocher dans la défausse, défausser sa carte dans la défausse). C'est également dans cette méthode que l'on va vérifier à la fin du tour du joueur si une colonne est supprimable, ou si le joueur a fini la partie (toutes ses cartes sont retournées), dans ce cas on appellera la méthode `finPartie()`.
- **joueurPioche()** : permet de faire piocher le joueur dans la pioche en appelant les méthodes `recupererHaut` et `supprimerHaut` de la classe Pioche et d'actualiser l'attribut `main` de deck (correspondant à la main du joueur) en conséquence. Main prendra donc la valeur de `recupererHaut`.
- **joueurDefausse()** : permet de faire défausser la carte contenue dans l'attribut `main` de l'attribut deck (de type Deck) du joueur. Ainsi, l'attribut `main` prendra la valeur null, son ancienne valeur sera mise dans la défausse avec la méthode `empiler()`.
- **joueurPiocheDefausse()** : permet de faire piocher le joueur dans la défausse en appelant les méthodes `recupererHaut` et `supprimerHaut` de la classe Defausse et d'actualiser l'attribut `main` de deck (correspondant à la main du joueur) en conséquence. Main prendra donc la valeur de `recupererHaut`.
- **demanderProchainJoueur()** : incrémente l'attribut `joueurActu` de 1 si elle peut, dans le cas contraire l'attribut prendra la valeur de 0 (on recommence un nouveau tour). Change également l'attribut `etat` de joueur en conséquence.
- **finPartie()** : cette méthode est appelée lorsque le nombre de cartes visibles est égal au nombre de cartes dans le deck du joueur qui vient de terminer son tour. La méthode permet de faire un dernier tour jusqu'au joueur qui a terminé et termine la partie.
- **donneTousLesPoints()** : cette méthode permet de donner les points de chaque joueur.
- **donneGagnant()** : renvoie le joueur qui a gagné la partie.
- **nouveauJoueur()** : crée une instance de Joueur ainsi qu'une instance de Deck qui correspondra à l'attribut deck de Joueur.
- **initialiseJeuCarte()** : initialise le jeu de cartes qui sera utilisé pour remplir les decks des joueurs ainsi que la pioche.
- **renouvellePioche()** : méthode permettant de renouveler le contenu de la pioche dans le cas où celle-ci serait vide. Pour ce faire, on utilisera la méthode de Defausse `donneDefausse()` et la méthode de Pioche `nouvellePioche`.

Classe Joueur:

Attributs:

- **nom** (String) : il s'agit du nom du joueur.
- **id** (Int) : l'id du joueur est l'index du joueur dans `listeJoueur`, un attribut de Partie.
- **etat** (String?) : permet d'avoir accès facilement à l'état du joueur, l'action qu'il réalise à un moment donné.
- **deck** (Deck) : instance de Deck spécifique au joueur.

Méthodes:

- **donneNom()** : renvoie le nom du joueur.
- **donneId()** : renvoie l'id du joueur.
- **donneEtat()** : renvoie l'état du joueur.
- **changeEtat(etat : String?)** : méthode appelée par la partie pour changer l'état du joueur.

Classe Deck:

Attributs:

- **main** (Carte?) : permet de stocker une carte piochée avant de la défausser ou de la placer dans le deck.
- **points** (Int) : score total affiché sur les cartes du deck.
- **nbreCartes** (Int) : nombre de cartes dans le deck.
- **nbreCartesVisible** (Int) : nombre de cartes retournées du côté visible dans le deck.
- **tableauCarte** (MutableList<MutableList<Carte?>>) : matrice de cartes formant le deck.

Méthodes:

- **retourneDeuxCarte()** : permet de retourner deux cartes en début de partie.
- **estSupprimable()** : renvoie les indices des colonnes supprimables.
- **supprimerColonne(colonnes : MutableList<Int>)** : supprime les colonnes du deck.
- **remplacerCarte(ligne : Int, colonne : Int)** : remplace la carte spécifiée par les coordonnées avec la carte dans la main, en appelant `modifierMain()` celle-ci devient la nouvelle main et sera défaussée ultérieurement.
- **comptePoints()** : compte les points affichés sur les cartes pour totaliser le score du joueur.
- **modifierMain(nouvelleMain : Carte?)** : change la main avec une carte spécifiée.
- **donneMain() : Carte?** : renvoie la carte dans la main.
- **compteNombreCarte()** : met à jour le nombre de cartes dans le deck.
- **compteNombreCarteVisible()** : met à jour le nombre de cartes dans le deck dont l'attribut visible est vrai.
- **verifieDeckRetourne() : Boolean** : compare le nombre de cartes retournées avec le nombre de cartes dans le deck.
- **donneNombreCarte() : Int** : renvoie le nombre de cartes dans le deck.

Classe Defausse:

Attributs:

- **pileCartes** (MutableList<Carte>) : MutableList de cartes dont on se sert pour représenter la défausse.
- **taille** (Int) : correspond à la taille de `pileCartes`.

Méthodes:

- **recupererHaut() : Carte** : renvoie la dernière carte ajoutée dans la défausse.
- **empiler(carte : Carte)** : empile la carte mise en paramètre dans la défausse.
- **supprimerHaut()** : supprime la dernière carte ajoutée dans la défausse dans le cas où le joueur pioche dedans.
- **donneDefausse() : MutableList<Carte>** : retourne le contenu de la défausse sans la dernière carte qui y a été ajoutée.

Classe Pioche:

Attributs:

- **fileCartes** (MutableList<Carte>) : MutableList de cartes dont on se sert pour représenter la pioche.
- **taille** (Int) : correspond à la taille de `fileCartes`.

Méthodes:

- **recupererHaut() : Carte** : renvoie la dernière carte ajoutée dans la pioche.
- **supprimerHaut()** : supprime la dernière carte ajoutée dans la pioche dans le cas où le joueur pioche dedans.
- **nouvellePioche(fileCartes : MutableList<Carte>)** : permet de renouveler la pioche dans le cas où elle serait vide. La pioche prend ainsi la valeur de la MutableList de cartes mise en paramètre.

Classe Carte:

Attributs:

- **valeur** (Int) : correspond à la valeur de la carte.
- **couleur** (String) : correspond à la couleur de la carte.
- **visible** (Boolean) : un booléen qui renseigne si la carte est visible (retournée) ou pas. Toutes les cartes ont comme valeur de base false pour cet attribut.

Méthodes:

- **donneCouleur() : String** : renvoie la chaîne de caractères correspondant à la couleur de la carte.
- **donneValeur() : Int** : renvoie l'entier correspondant à la valeur de la carte.
- **estVisible() : Boolean** : renvoie un booléen, true si la carte est visible, false dans le cas contraire.
- **changeCarteVisibilite()** : met l'attribut visible de la carte à true.

Diagramme de séquence du début de partie :

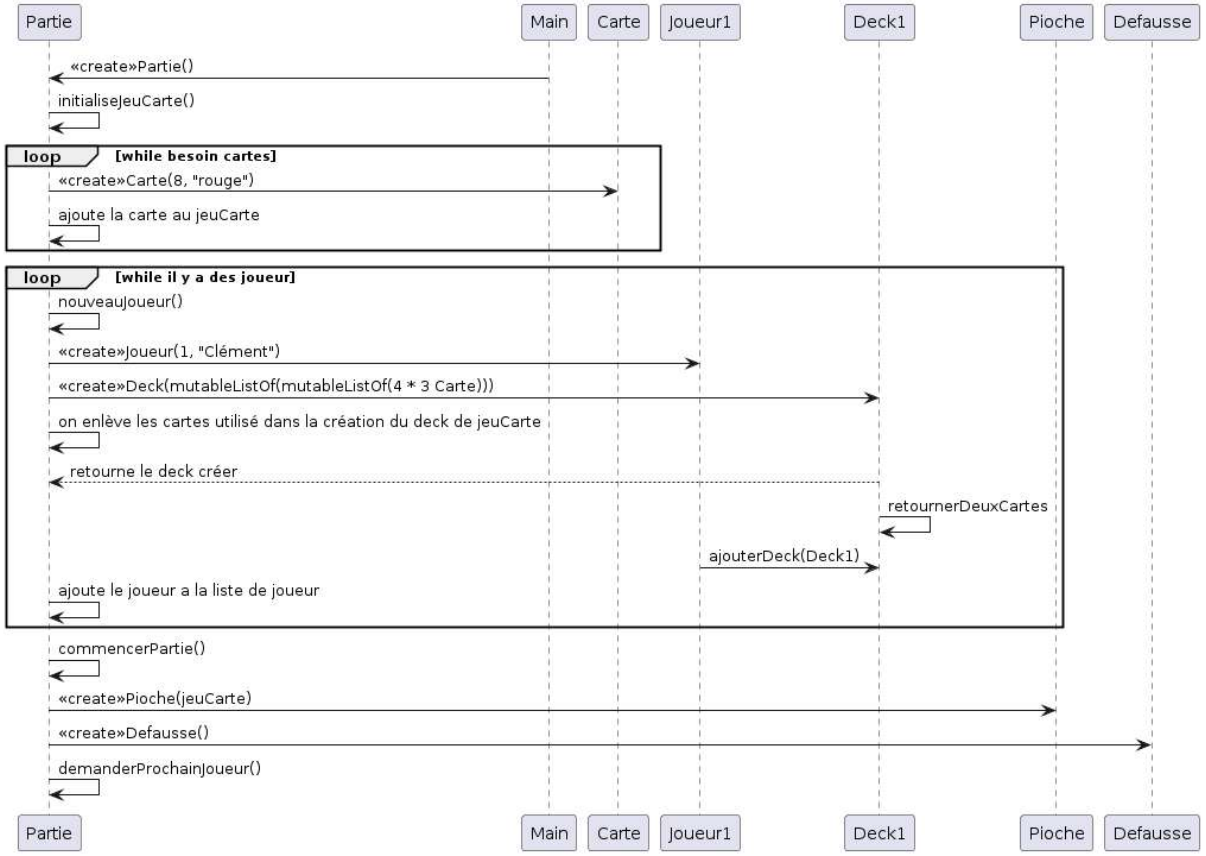


Diagramme de séquence de la pioche avec les deux choix différents:

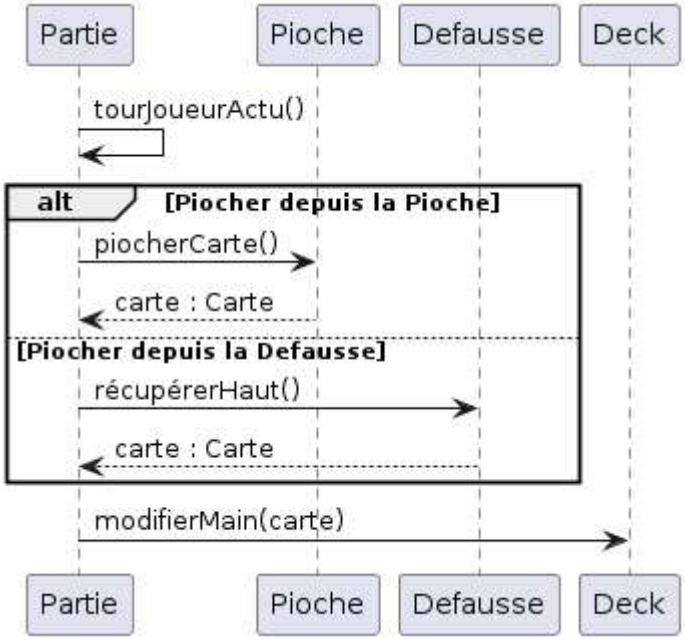


Diagramme de séquence de fin de tour

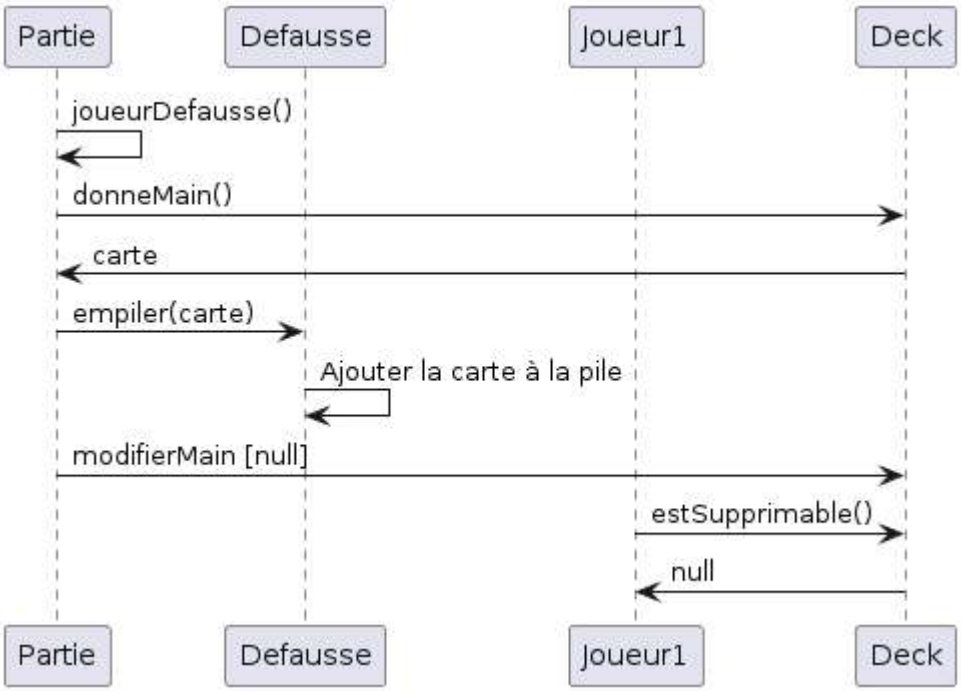


Diagramme de séquence de fin de partie